International Journal of Innovations in Engineering and Technology (IJIET)
http://dx.doi.org/10.21172/ijiet.84.009

Simulation of a Simple Bio-Mimetic Robot with Neuromorphic Control System and Optimization Based on the Genetic Algorithm

Bauyrzhan Ospan
Department of Information Systems
Kazakh University of Technology and Business, Astana, Akmola state, Kazakhstan

Abstract- A simulation of the autonomous differential wheeled mobile robot with a survival intellect was organized in virtual environment created in Matlab and Simulink software. A feed forward two-layer artificial neural network was used as control algorithm for path planning of the simple robot. While the simulations two different optimizing algorithms were implemented to adjust the synapses of the artificial neural network, which are linear and genetic algorithms. The analysis of the simulations demonstrates that the heuristic-based genetic optimization algorithm has better results than a linear optimization. The results of simulations can be used in hardware implementation of a simple mobile robot with neuromorphic control system.

Keywords – Artificial Neural Network, Genetic Algorithm, Simulation, Simple Robot, Matlab, Simulink, Survival Intellect

I. INTRODUCTION

One of the main objectives in Robotics is the development of autonomous robots. Autonomous robots accept high-level descriptions of tasks and execute them without operator control. The task descriptions contain what the user wants to be done rather than instructions how to do it.

Autonomy and intelligence are two different concepts. However, to be autonomous a robot must have basic survival intellect, introduced by Hasslacher and Tilden [1]. Inspired by biological creatures, they declaimed three rules that guarantee a robot's survival. First, a machine must protect its existence. Second, a machine must gain more energy that it consumes. Third, a robot has to move to the goal [1].

The research on autonomous robots includes number of topics, but one of the most important fields is a path planning. In this paper, I will define path planning as determination of an optimal collision-free path from start to goal point or goal points [2].

My primary research interest is a local path planning [2]. The local path planning is a task for the mobile robot to get information through sensors, to analyze data and to adapt to the environment, when it does not know the location of obstacles [2].

This work can be considered as a continuation of the project of Dr. Michele Folgheraiter, and his colleagues from Politecnico di Milano [3],[4]. In this work I improved and modified the neural network algorithm developed by Folgheraiter et al. [3][4], and introduced optimization algorithm.

Returning to the three survival laws of biomorphic robots, robot obeys all three of them. First, it protects its existence by avoiding obstacles. Second, it monitors energy level and changes its behavior depending on it. Third, the robot tries to reach its destination points that are energy station and reproduction station in an efficient way and learns to optimize its path with each iteration.

In the following sections, a current progress of research in the discipline of path planning will be described, followed by a description of the neural network model and simulation results. Finally, I will conclude the report with a summary of results and suggestions on future works.

II. LITERATURE REVIEW

There is a significant amount of research in the field of robot path planning. According to Tang, Khaksar, Ismail, and Ariffin [5], the discipline was launched in 1960s, but it has obtained attention of the researchers only two decades

later, after Lozano and Wesley had published their article on spatial planning. Having analyzed papers in the path planning field from 1980 to 2010, Tang et al. [5] categorized current methodology into two groups: deterministic and heuristic-based. Heuristic approaches include Neural Networks, Genetic Algorithms, Simulated Annealing, Ant Colony Optimization, Particle Swarm Optimization, Stigmergy, Wavelet Theory, Tabu Search, Fuzzy Logic [5].

One of the popular methods in heuristic group is Neural Network that takes 12.75% of approaches used in robot motion planning [5]. Imitating the brain's neurons, Artificial Neural Networks (ANNs) are the models based on a network of simple processing elements called nodes. Repeatedly solving a problem and using stored experiential knowledge of previous attempts - in other words, through learning process - ANN algorithm improves its performance by adjusting the network's adaptable nodes [6].

Folgheraiter et al. [3],[4] developed their own neural network architecture. Their neural control system is similar to the one that was used in this paper with several modifications and will be described in the following sections.

As for the future work, Folgheraiter et al. [3],[4] also suggested developing a performance-optimizing algorithm that adjusts the synapses. In this project, I introduce two training algorithms that run in simulation and use superposition of time and energy level (spent for reaching the target) as an error function.

III. Neural Control System

The artificial neural network is a feedforward two-layer network. As shown in the Fig. 1, the first layer is composed of 7 neurons that are connected to inputs from 7 sensors:

1. 2 light sensors (LS Energy R,L) that are responsible for detecting a target. Each light sensor is connected to a separate neuron (neurons N1-3, N1-4);

2. 2 light sensors (LS Target R,L) that are responsible for detecting an energy station. Each light sensor is connected to a separate neuron (neurons N1-2, N1-5);

3. 2 ultrasonic sensors (US R,L) that serve as the bumper sensors. Each ultrasonic sensor is connected to a separate neuron (neurons N1-1, N1-6);

4. 7th neuron N1-7 takes input from an energy level monitor.

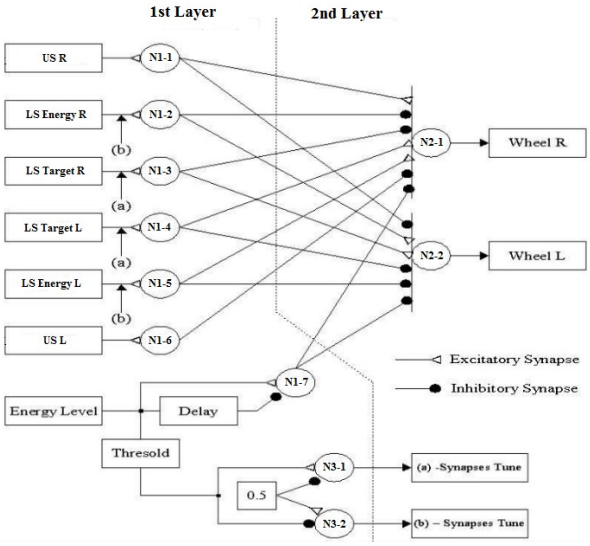

Figure 1. The Neural Controller Architecture

The main changes from the original architecture are the following: first, in this model ultrasonic sensors serve as contact sensors, and, second, light sensors are used instead of sound sensors.

The second layer is composed of two neurons, each controlling one motor. Neuron N2-1 controls the right wheel, while neuron N2-2 controls the left wheel.

At this stage of the project, the robot's overall behavior consists of four separate specific behaviors. The robot's main task is to (1) reach a target point, while (2) avoiding collisions with obstacles. It (3) monitors its energy level

during task's completion, and when the energy level falls below a specific threshold value, the robot changes its plan and (4) heads for the energy station instead.

Reaching a Target: in order to detect the target, the robot uses two light sensors, which are neurons N1-3 (right sensor) and N1-4 (left sensor). N1-3 realizes inhibitory synapse with N2-1 neuron and excitatory synapse with N2-2 neuron. Symmetrically, N1-4 realizes inhibitory synapse with N2-2 neuron and excitatory synapse with N2-1 neuron. Such connections of the sub-network allows motor to turn left if it receives stronger signal from the left light target sensor, and right if it receives stronger signal from the right side.

Collisions Avoidance: in order to detect the obstacles, the robot uses two ultrasonic sensors, which are represented by neurons N1-1 (right sensor) and N1-6 (left sensor). N1-1 realizes excitatory synapse with N2-1 neuron and inhibitory synapse with N2-2 neuron. However, N1-6 neuron realizes only inhibitory synapse with N2-1 neuron, thus making the sub-network asymmetric. This high priority of the right motor in making decisions is used to escape the situation when the obstacle is right in front of the robot. Such connections of the sub-network allows motor to turn left if it receives stronger signal from the right ultrasonic sensors and right if it receives stronger signal from the left side. Thus, the sub-network produces the opposite behavior than that of the previously described sub-network of reaching a target. The value of synapses determines the speed of turns.

Energy Level Monitoring: this sub-network involves neurons N1-7, realizing inhibitory synapses with N2-1 and N2-2, and neurons N3-1, N3-2, which influence the synapse values of neurons from N1-2 to N1-5. This part of the network plays a key role in the controller, because it dictates which behavior will take control of the motors: reaching the target or reaching the energy station. When the energy level falls below the fixed threshold value – in the system it is 50% - a signal reaches both excitatory synapse of N3-1 and inhibitory synapse of N3-2. Thus, the behavior "Reaching the Energy Station" dominates, when N3-1 is activated and N3-2 is deactivated, correspondingly. The more robot needs energy, the more it prefers reaching the energy station to reaching the target. When the robot, eventually, reaches the energy station, the positive change of energy level is noticed by neuron N1-7, and, as a result, the robot stays motionless until it is charged to 95%. It should be noted, that this sub-network does not influence the neurons N1-1 and N1-6, responsible for avoiding the obstacles.

Reaching the Energy Station: this behavior is identical to the behavior of reaching the target, except that it uses different light sensors and, therefore, involves different set of neurons (N1-2, N1-5, N2-1, N2-2).

Eqn. 1 describes each neuron in controller, while eqn. 2 describes the activation function. In the equations, P is membrane potential, Y is neuron's output, x_i and W_(e_i)- excitatory inputs and its weights, x_j and W_(e_j)- inhibitory inputs and its weights, k - dynamics constant (higher values mean faster reaction to input changes) [3],[4].

$$\begin{cases} \dot{P} = k\left(\sum_{i=1}^{n} W_{e_i} x_i - \sum_{j=1}^{m} W_{e_j} x_j - P\right) \\ Y = Th(P) \end{cases} \quad (1)$$

As an activation function, I use a piece-wise linear function, instead of non-linear one (i.e. sigmoid), because the primary goal is to avoid saturation. Forgetting mechanism, represented by the term -P in eqn. 1, allows a neuron to avoid saturation [3],[4].

$$Th(P) = \begin{cases} 0: & P \le 0 \\ P: 0 < & P \le 1 \\ 1: & P > 1 \end{cases} \quad (2)$$

Although neurons from N1-2 to N1-5 have similar model, they have additional input coming from neurons N3-1 and N3-2. This influence of neurons N3-1 and N3-2 on mentioned neurons can be modeled with eqn. 3, where W_s is synapse internal state and W_c is the tuning signal coming from N3 type of neurons [3],[4].

$$\begin{cases} W_s = W_c - k_d \\ W = Th(W_s) \end{cases} \quad (3)$$

IV. Simulation

To simulate the neural controller behavior and tune the weights of the neurons I have developed a virtual world as a runtime function for the Simulink model based on the work of the Dr. Folgheraiter et al. [3],[4].

The first group of functions is responsible for calculations done by the neural network.

The second group of functions creates the virtual arena with walls, random obstacles, one target point, and one energy station.

The third group of functions calculates the input values of the energy station and the target point sensors. Sensor inputs are not calculated as a linear dependence on distance between sensors and destination points, but as a quadratic polynomial, thus approximating the physical characteristics of light.

The fourth group of functions generates input signals from ultrasonic sensors in a form of quadratic polynomial, which is dependent on the distance between the sensors and the nearest object. Moreover, they are activated only if the distance between sensors and obstacles is less than activating distance (equal to 20 cm).

The last group of functions is responsible for the kinematic behavior of the robot in the virtual environment. Robot motions are governed by the equations of differential drive configuration. In the calculations, I take into account the inertia of the wheels and friction forces, like air resistance. By calculating friction forces, I am able to calculate the energy spent for the motion.

Combining all the functions, including additional visualizations functions not described here, I am able to manually tune neurons weight parameters and simulate the behavior of the neural controller.

V. Manual Tuning

First, I started the manual tuning with the target point neuron weight on the Fig. 2. I eliminated effects of other neurons by deleting the energy station and obstacles from the environment. Similarly, I tuned the energy station neuron weight and found that the behavior of the target and energy station neuron is identical.

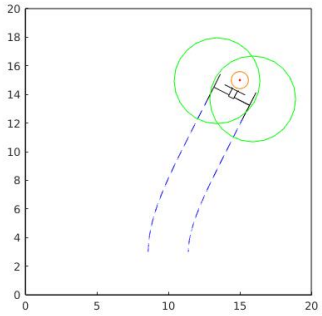

Figure 2. Manual Tuning Of The Target Point Neuron Weight

Then I tuned the bumper actuating radius to make the robot move effectively and avoid the obstacles. I have decreased the actuating distance to avoid the deadlock situations on the Fig. 3.

Third, I tuned the bumper neuron weight by creating one obstacle between the energy station and the starting point. I came to the conclusion that in order to make robot avoid the obstacles, the weight of the bumper neurons has to be at least twice larger than that of the energy station neuron.

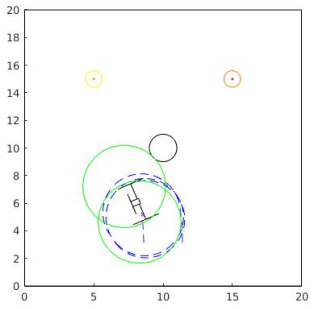

Figure 3. Deadlock Situation Due To Bumpers' High Activation Distance

After a number of simulation cycles, I have obtained the most suitable weights of the neurons for this particular environment on the Fig. 4. However, if environment is changed, similar procedures should occur again. Therefore, I

came up with an idea to create a training algorithm that will automatically adjust the weights of neurons based on some cost function.

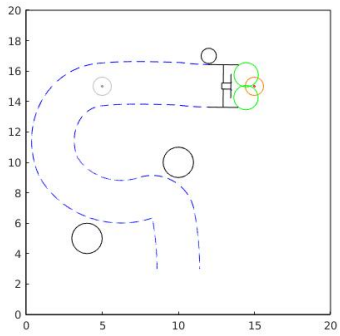

Figure 4. Final Result Of The Manual Tuning

VI. Cost Function

To increase performance of the system I developed an adaptive element of the control unit represented by a training algorithm. To develop the training algorithm I constructed complex environment with one energy point, one target point, and three randomly located obstacles - the environment purely identical to the one from the manual tuning part.

Before introducing the algorithm, I need to define the cost function. The first idea was to use energy consumption as a cost function. After several iterations, the robot learned that the best way to keep energy is to stay motionless and, as a result, the time spent for reaching the target went to infinity. The second idea was to use the time spent for reaching the target as the cost function instead. This approach also produced inappropriate results, in which energy consumed was neglected. Thus, I concluded that the best cost function would be the linear combination of time (T) and energy (E) in the eqn. 4. In other words, I wanted to find an optimal trade-off between energy consumed and time spent:

$$F_{cost} = k_1 * E + k_2 * T \quad (4)$$

In order to understand the nature of the cost function, I plotted several cost functions for various combinations of k1/k2 and found that the optimal ratio should be around 40. Fig. 5 is a plot of the cost function in 3D space, where x-axis is the weight of target neurons, and y-axis is the weight of the bumper neurons.

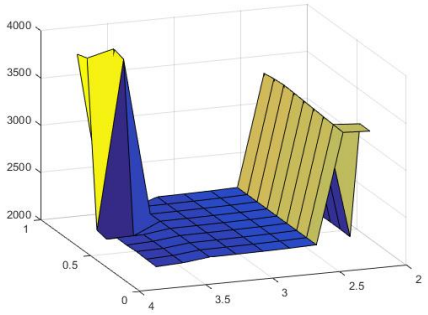

Figure 5. Surface Plot Of The Cost Function

VII. Linear Algorithm

Linear algorithm is based on three empirically derived assumptions:

1. The inhibitory weight of the neuron pair has to be negative, and in absolute values less than the excited neuron weight, which is the constant value;
2. The weight of the energy station neurons and the weight of target point neurons is the same;

3. There is no correlation between the bumper neurons weight and the target neurons weight.

In fact, the last assumption is a basis for linear algorithm (Fig. 6).

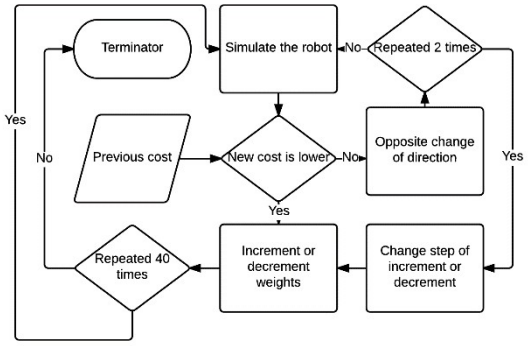

Figure 6. Linear Algorithm

The algorithm works like standard number search program. First, I create a nominal robot with weight values found by manual tuning. Then the trainer increments or decrements weight value of one neuron by a particular step. It calculates the resulting value of the cost function and compares it with the previous one. If the value is lower, the script continues to change the weight in the same direction; otherwise, it changes the sign of the step, and follows the procedure of comparing the values of two cost functions. If the sign changes twice in a row, it means that the optimal value is between these two cost functions, and, therefore, I reduce the step value and continue the procedure. The step size starts at 0.1 and ends at 0.001. After I find the optimal value for one weight, I repeat the procedure with the second weight. The trainer terminates after 40 iterations.

After the training, I found that the linear algorithm does not work properly. Despite the correctness of the algorithm's logic, the algorithm does not produce effective value of the cost function. As a result, I had to use a different algorithm that suits for nonlinear system.

VIII. GENETIC ALGORITHM

In the Genetic Algorithm, the weights of the neural network can be considered as genes. The progenitor or zero generation species is the robot with manually tuned weights. This robot gives a birth for a 36 robots with mutated genes. In that case, mutation means the change of weight value by 0.1, 0.01, and 0.001 in both positive and negative directions, which explains why I have 36 species in the next generation. At each generation, the trainer evaluates the cost functions of each robot and chooses the robot with the lowest cost function value. Then, this robot mutates in similar manner, and the procedure repeats until the 20th generation. Due to having only two genes, one cost function, and limits of the gene mutation, I skip the crossover procedure.

Genetic Algorithm reached the effective minimum point, which is lower than that achieved by Linear Algorithm, and significantly faster in terms of number of iterations/generations in the Fig. 7. On the other hand, in terms of training time, the Genetic Algorithm is slower due to the huge population of each generation. In the plot, y-value represents the cost function value, while x-axis represents the iterations/generations.

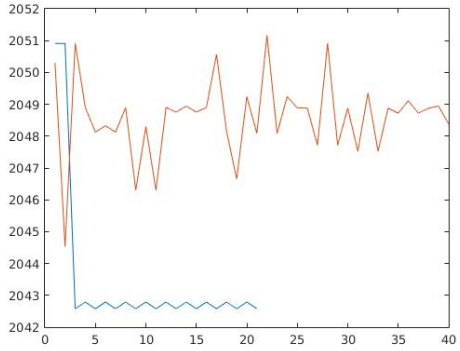

Figure 7. Comparison Between Cost Functions Of Genetic Algorithm (Blue) And Linear Algorithm (Orange)

Fig. 8 demonstrates the best result of the neural network with genetic optimization algorithm.

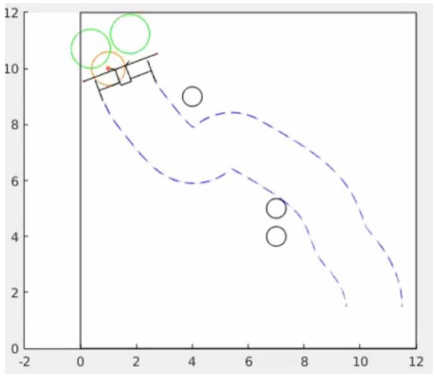

Figure 8. Simulation With Genetic Algorithm

IX. Conclusion

In order to develop adaptive control for mobile robots, I implemented a neural network and developed two training algorithms. According to Zou et al., the best results in real-world applications, however, are achieved by hybrid approaches that use a combination of Neural Network and other heuristic-based methods [6]. In this simulation, I tried to improve the robot's performance with Genetic Algorithm. Nevertheless, as for the future work I suggest to integrate other AI-based methods into the system.

X. Acknowledgments

I would like to thank Dr. Michele Folgheraiter, for his guidance and help during the completion of the project, as well as for his expertise shared with me. Without him, this project would not have been done.

REFERENCES

[1] B. Hasslacher, M.W.Tilden (1994) Living Machines: 17-19.
[2] P.Raja, S.Pugazhenthi (2012) Optimal path planning of mobile robots: A review. International Journal of Physical Sciences, 7(9): 1314-1320.
[3] M.Folgheraiter, G.Gini, A.Nava, N.Mottola. A Bioinspired Neural Controller for a Mobile Robot.
[4] M.Folgheraiter, G.Gini, A.Nava, V.Lumare (2007) CrickBot: A mobile robot with bio-mimetic control architecture.
[5] S.H.Tang, W.Khaksar, N.B.Ismail, M.K.A.Ariffin (2012) A Review on Robot Motion Planning Approaches. Pertanika J.Sci. & Technol, 20(1): 15-29.
[6] A.Atyabi, D.M.W.Powers (2013) Review of classical and heuristic-based navigation and path planning approaches. International Journal of Advancements in Computing Technology, 5 (14).